EE 107 Intro. to ECET with software development, Spring 2020

Department of Electrical and Computer Engineering & Technology, MNSU
Dr. Qun Zhang

Handout 2 most related to Chapters 3 of book

Key points covered

0. Review: numbers of codes (brief)
   binary codes (recall amplifier abstraction and NOT gate); Source coding and ASCII code ('a' as binary 01100001); number representation (unsigned and signed); 2's complete numbers

Term	Definition
bit	0 or 1
byte (B)	a group of 8 bits
nibble (nybble)	half a byte (4 bits)
word (w)	a group of bits that is processed simultaneously. a word may consist of one or multiple bytes machine dependent (ex: 8086 – 16 bits; 80386/80486/Pentium – 32 bits) (nowadays – i5/i7 processors – 64 bits)
double word	2 words
msb (most significant bit)	the leftmost bit in a word
lsb (least significant bit)	the rightmost bit in a word
Hz (hertz)	reciprocal of second

Term	Normal Usage	Usage as a Power of 2
Kilo (K)	10^3	2^{10} =1,024
Mega (M)	10^6	2^{20} =1,048,576
Giga (G)	10^9	2^{30} =1,073,741,824
Tera (T)	10^{12}	2^{40} =1,099,511,627,776
Mili (m)	10^{-3}	
Micro (m)	10^{-6}	
Nano (n)	10^{-9}	
Pico (p)	10^{-12}	

1. For a new programmer to be successful in this course: you will need work hard in those : Problem solving; master knowledge of concepts; develop programming skills (practice!). Now we start with an example of problem solving …

2. Recall: how to develop a computer program to compute $\sqrt{2}$? what algorithms you can come up, let's assume C PL can only handle ADD, SUB, Multiplication, and Division.

Another example: guess an integer number between 0 and 1000, how fast you can get the number by Q (is the number I guess less than your number?) /As?

a. Based on our algorithm, what will be required?

- represent knowledge with data structures (speak of data – we need data types)
  - (C is a strong typed language) (A type defines a set of values (*domain of the type*) and a set of operations that can be applied on those values)
- iterations and branching
- abstraction of procedures and organize and modularize systems (later on – functions/methods)
- algorithms and their analysis (e.g. complexity).

One key advice is to come up with algorithm and coding/programming style in practice that can be easy to understand by people – with some concise comments, everyone who read the code (program) can understand it.

3. C provides a set of primitive data types and operations (Chapter 3).
   …. Such as **void (standard)**, **float (always signed)**, **int**, short, long, long long, _Bool, **char**, _Complex and _Imaginary (C99) … (note other derived types such as pointer, array, structure will be discussed later).

   …. Such as +, -, *, /, % (modulus), >=, ==, != …

   So by analogy, in English or Chinesse we have words, in C, we have numbers, strings, simple operators.

   Remark: void - Has no values; Has only one operation: assignment

4. Expressions are **legal combinations of primitives** in any PL, including C. (Chapter 3) (Syntax). Examples of syntax errors
   English: Tom Kate. vs. Tom helps Kate.
   C PL: C=A b; vs. C=A*b;

   Remark about use of **variables** and their **declaration**:
   Variables are named memory locations that have a 1). Type (e.g., int, char, and consequently a size) 2). Identifier (name; follow rules in L#4 Slides 21, 22), 3). Value.

   Each variable in the program must be declared and defined!
   Declaration: to name a variable
   Definition: to create a variable, to reserve memory for it
   Usually a variable is declared and defined at the same time!

   C allows multiple variables of the same type to be defined in one statement

5. Expressions and computtations have values and meanings in any PL including C (Chapter 3). (Semantics).

    a. Static semantics: Syntactivally valid strings have meaning

       English: "He run" syntactically valid but static semantic error
       ◦programming language: 5*5 vs. 3 +"hi" (static semantic error)

    b. no semantic errors but different meaning than what programmer intended: NLcoeff=8/9*0.0002;

    Another good example: you like to calculate something you expect program will stop in 10 minutes; but the program runs for ever (infinite loop).

6. Constants: Constants are data whose values can not be changed while the program is running.

    Constants have a type (like variables)
    a. Integer constants
    b. Floating point constants
    c. Character constants
    d. String constants

    6.1. Integer constants are simply coded as you would use them in everyday life!
    The default type is signed integer or signed long integer if the number is large.
    You can override the default by specifying u or U (for unsigned) and l or L (for long) after the number.

    Examples:

-32271L	long int
-100	int
78	int
76542LU	unsigned long int

    Note: you cannot specify a short int (so don't think of specifying it).

    6.2. These are numbers with decimal parts.
    The default form is double.
    If you want the resulting type to be float or long double use F or L correspondingly.

    6.3. Character constants are enclosed between two single quotes. In addition, there can be a backslash \ (the escape character). The backslash is used when the character does not have a graphic associated with it.

    backspace: '\b'
    newline: '\n'

tab: ‘\t’
single quote: ‘ \ ‘ ’
double quote: ‘\”’
null character: ‘\0’

6.4. String constant: A sequence of characters enclosed in double quotes. “Hello, world!\n”

Remark: Difference between the null character and the empty string.
‘\0’ is null character
“ ” is empty string

Remark: Contants are used in program via three different ways:

- Literal: an unnamed constant used to specify data
- Defined: use the preprocessor command #define name expression (the expression that follows the name in the command replaces the name wherever it is found in the source program)
- Memory: use a C type qualifier: const type identifier = value; (memory constants fix the contents of a memory location)

7. Tons of notes for expression: imperative etc. … We move those to the next handout.

8. Learning by doing – programs to study (always pay attention to memory and scope).
Example 1.

```
#include <stdio.h>

int main (void)
{
    int       integerVar = 100;
    float     floatingVar = 331.79;
    double    doubleVar = 8.44e+11;
    char      charVar = 'W';

    _Bool     boolVar = 0;

    printf ("integerVar = %i\n", integerVar);
    printf ("floatingVar = %f\n", floatingVar);
    printf ("doubleVar = %e\n", doubleVar);
    printf ("doubleVar = %g\n", doubleVar);
    printf ("charVar = %c\n", charVar);

    printf ("boolVar = %i\n", boolVar);

    return 0;}
```

Example 2.

// Illustrate the use of various arithmetic operators -- **Review** addition; subtraction.

```
#include <stdio.h>

int main (void)
{
    int a = 100;
    int b = 2;
    int c = 25;
    int d = 4;
    int result;

    result = a - b;       // subtraction
    printf ("a - b = %i\n", result);

    result = b * c;       // multiplication
    printf ("b * c = %i\n", result);

    result = a / c;       // division
    printf ("a / c = %i\n", result);

    result = a + b * c;   // precedence
    printf ("a + b * c = %i\n", result);

    printf ("a * b + c * d = %i\n", a * b + c * d);

    return 0;
}
```

Example 3.

```
// More arithmetic expressions

#include <stdio.h>

int main (void)
{
    int   a = 25;
    int   b = 2;

    float c = 25.0;
    float d = 2.0;

    printf ("6 + a / 5 * b = %i\n", 6 + a / 5 * b);
    printf ("a / b * b = %i\n", a / b * b);
    printf ("c / d * d = %f\n", c / d * d);
    printf ("-a = %i\n", -a);

    return 0;
}
```

Example 4.

```
// The modulus operator

#include <stdio.h>

int main (void)
{
     int a = 25, b = 5, c = 10, d = 7;

     printf("a = %i, b = %i, c = %i, and d = %i\n", a, b, c, d);
     printf ("a %% b = %i\n", a % b);
     printf ("a %% c = %i\n", a % c);
     printf ("a %% d = %i\n", a % d);
     printf ("a / d * d + a %% d = %i\n",
               a / d * d + a % d);

     return 0;
}
```

Example 5. (casting).

```
// Basic conversions in C

#include <stdio.h>

int main (void)
{
    float  f1 = 123.125, f2;
    int    i1, i2 = -150;
    char      c = 'a';

    i1 = f1;               // floating to integer conversion
    printf ("%f assigned to an int produces %i\n", f1, i1);

    f1 = i2;               // integer to floating conversion
    printf ("%i assigned to a float produces %f\n", i2, f1);

    f1 = i2 / 100;         // integer divided by integer
    printf ("%i divided by 100 produces %f\n", i2, f1);

    f2 = i2 / 100.0;       // integer divided by a float
    printf ("%i divided by 100.0 produces %f\n", i2, f2);

    f2 = (float) i2 / 100;     // type cast operator
    printf ("(float) %i divided by 100 produces %f\n", i2, f2);

    return 0;
}
```

Remark: please refer to the langue specs in Appendix of the book, including the precedence orders.